RDF Modelling and SPARQL Processing of SQL Abstract Syntax Trees

Corentin Follenfant, Olivier Corby, Fabien Gandon, David Trastour

▶ **To cite this version:**

Corentin Follenfant, Olivier Corby, Fabien Gandon, David Trastour. RDF Modelling and SPARQL Processing of SQL Abstract Syntax Trees. PSW - 1st Workshop on Programming the Semantic Web, Nov 2012, Boston, United States. hal-00759034

HAL Id: hal-00759034

https://inria.hal.science/hal-00759034

Submitted on 29 Nov 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

RDF Modelling and SPARQL Processing of SQL Abstract Syntax Trees

Corentin Follenfant[1,2], Olivier Corby[1], Fabien Gandon[1], and David Trastour[2]

[1] INRIA Sophia Antipolis Méditerranée
firstname.lastname@inria.fr
[2] SAP Research
firstname.lastname@sap.com

Abstract. Most enterprise systems rely on relational databases, and therefore SQL queries, to populate dynamic documents such as business intelligence reports, dashboards or spreadsheets. These queries represent metadata about the documents, thus they can feed information retrieval systems such as recommender systems or search engines. In this paper we propose to automatically annotate documents with structured representations of their SQL queries expressed with RDF graphs. We show that SPARQL is a natural language to query these SQL queries, i.e. to perform meta-querying, and discuss challenges that arise from this approach.

Keywords: query modelling, query languages, query graphs, abstract syntax trees, Linked Enterprise Data

1 Introduction

The vast amounts of data federated into organisations' data warehouses are usually analyzed through a variety of dynamic documents authored with business intelligence (BI) tools. Consumption of such documents is accessible to business users as the data is organized into graphs, charts or pivot tables. However, their authoring remains limited to users with technical skills, as it often involves query editing. A broad range of information retrieval (IR) systems have proved useful in assisting users by suggesting them similar documents or queries for reuse, but they depend on consistent, i.e. human-authored and thus sparse metadata. The sole formal common ground shared by documents from differents tools is their SQL or derivatives (e.g. MDX) queries.

We consider these SQL queries as metadata that IR systems can leverage with two desirable features: search and modification. *Query search* can be used to find queries that target a specific rowset or apply given transformations onto it, by looking at both queries content and structure. For instance, one can look for SQL queries that perform an aggregation of `Order Price` over `Customers`. A search engine shall address several problems to find queries that match this pattern. First, it must identify the query atoms that refer to `Order Price` and `Customers`, likely columns from an `Orders` table, and align their different occurrences (e.g. aliases). Second, it must consider the position of these atoms with respect to

the scope of given operators, in this case an aggregation function and a `GROUP BY` clause. Finally, it has to look for all the operators considered as aggregate functions, i.e. it needs rich typing of the query structure. *Query modification* is another IR approach which attempts to edit a query in order to have it yield more results (relaxation), less results (restriction), different results (shift), or equivalent results with better performance (rewriting). As it involves editing the target query, the modification approach requires a read/write meta-querying language. It can be used to search for similar queries that do not exactly match the initialy provided query pattern.

In this paper we present an approach where we annotate BI documents with their SQL queries. We represent SQL queries with their abstract syntax trees (ASTs) as RDF graphs, which allow 1) to type the tree nodes accordingly to the SQL construction they represent, 2) to identify queries with the URI of a named graph containing the graph of their AST, and 3) to annotate queries with contextual metadata, e.g. the documents that trigger them. We then present SPARQL queries that can be used against a repository of such SQL queries modelled in RDF, i.e. to perform meta-querying according to the desirable features earlier described.

The remainder of this paper is organized as follows: in section 2 we present related works that frame ours. Section 3 then details the framework for modelling SQL queries with RDF graphs, and outlines a vocabulary that captures basic SQL ASTs. We then detail SPARQL queries usable for convenient search and modification of SQL queries expressed with our model, with preliminary examples shown in section 4, before concluding and giving perspectives in section 5.

2 Related Work

The task of authoring BI documents can be made easier by IR systems that take into account metadata about their queries [10], which Priebe and Pernul call *query context*. However, this refers to "elements", i.e. business entities shown as a result of the query execution, rather than the fine-grained query structure underlying the document.

Modelling and storing queries to enable meta-querying has been investigated in both RDF and relational formalisms, but to the best of our knowledge no approach relates them in either way. In the relational world SQL queries can be broken down into XML trees, thus requiring additional XML querying operators to be added to SQL meta-queries [2]. SnipSuggest [7] aims at user assistance through query autocompletion mechanisms. It maintains a query repository in relational tables where query metadata are stored as features, where we consider the whole query AST. Relational query rewriting has been approached from a graph perspective by decomposing queries into atomic subgoals to be connected [9]. The main query modelling contribution in the semantic web world lies in the SPIN vocabulary [8], which allow SPARQL queries to be represented in RDF.

Ferré proposed to represent ASTs of generic mathematical expressions as Prolog terms, which are easily mapped in RDF. He focuses on search and

navigational features provided through an extension of LISQL [5]. While we merely extend this model to capture SQL queries ASTs, we process them with standard SPARQL 1.1.

Besides query modelling, works aimed at connecting relational and semantic web applications have rather focused on translating data and queries from one formalism to the other, e.g. from relational to RDF with the R2RML standard currently developed by W3C [4], and vice versa.

3 Query Modelling and Semantic Web Graphs

The same way SPIN captures SPARQL queries into RDF [8], we want to capture SQL queries in RDF. SQL queries can be represented with abstract syntax trees, when looking at all the constructs of a query as expressions in prefix notation. Thus, a set of queries can be represented with an RDF graph by appropriately minting URIs for shared SQL identifiers [4]. We model the syntactic structure of the trees, and RDF allow to further annotate them with additional metadata or link them together. For example, references to cryptic column names that abound in enterprise software can be annotated with the concepts or attributes types they represent.

The rationale for capturing SQL ASTs in RDF is the following: all the constructs are composed of typed blank nodes that represent operators and operands, in a broad sense. Operators include SQL clauses, mathematical terms, logical predicates or functions. Whatever the arity of an operator, its operands are represented as `rdf:List` collections (closed containers). The arguments are attached to the typed node with the `argument` property. Operands themselves can be atoms or further operators. Atoms are either constants with a `rdf:value` property, or SQL objects with a `sql:identifier` property, and have no list of arguments. The RDFS vocabulary used to type the nodes of ASTs according to this is outlined in figure 1. To keep the diagram concise we omit leaf subclasses that denote proper functions (mainly aggregates sum, average, min/max), logical predicates and mathematical operators. We use an arbitrary namespace with the `sql:` prefix.

This extract is limited to the selection subset of the data manipulation language (DML), but can be extended to capture other DML as well as data description language statements.

Listing 1 shows an example SQL query retrieving the total amount of orders per customer which were placed in Q3. This query matches the search pattern described in section 1.

```
1 SELECT Customer, SUM(OrderPrice)
2 FROM Orders
3 WHERE OrderQuarter = 'Q3'
4 GROUP BY Customer
```

Listing 1: Sample SQL Query Q1

We derive the AST of this query as shown in figure 2. The tree model implies that all the siblings of an AST node represent arguments in the scope of the

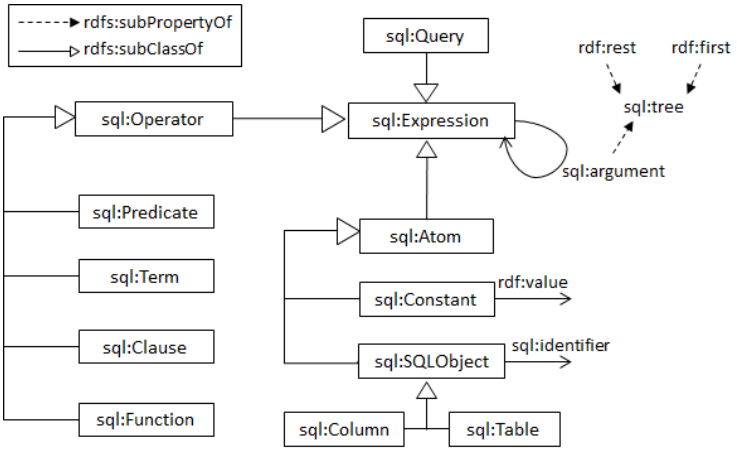

Fig. 1: Outline of SQL AST vocabulary

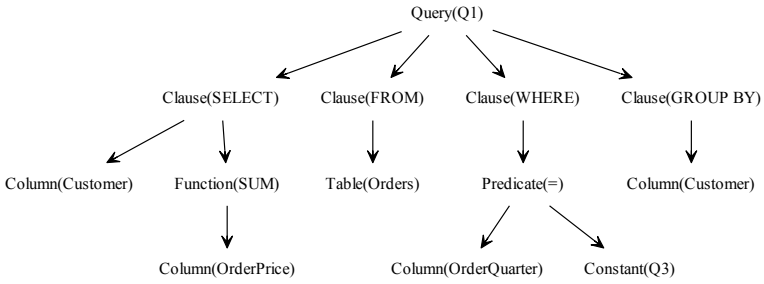

Fig. 2: AST for Q1

current operator, whether there are atoms, subexpressions, scalar or aggregate functions, or subqueries. Listing 2 presents the query Q1 modelled in RDF, by applying the process earlier described. The types `sql:Sum` and `sql:Equals` are respectively subclasses of `sql:Function` and `sql:Predicate`; while the types representing clauses are subclasses of `sql:Clause`.

```
1  [] a sql:Query ;
2    sql:argument (
3       [ a sql:Select ;
4         sql:argument (
5           [ a sql:Column ; sql:identifier "Customer" ]
6           [ a sql:Sum ; sql:argument (
7               [ a sql:Column ; sql:identifier "OrderPrice" ] ) ] ) ]
8       [ a sql:From ;
9         sql:argument (
10          [ a sql:Table ; sql:identifier "Orders"] ) ]
11      [ a sql:Where ;
12        sql:argument (
13          [ a sql:Equals ; sql:argument (
```

```
14              [ a sql:Column ; sql:identifier "OrderQuarter" ]
15              [ a sql:Constant ; rdf:value "Q3" ] ) ] ) ]
16      [ a sql:GroupBy ;
17        sql:argument (
18          [ a sql:Column ; sql:identifier "Customer"] ) ] ) .
```

Listing 2: Q1 modelled in RDF/Turtle

4 Preliminary Evaluation: SPARQL Meta-Querying

Meta-querying is defined as querying a database containing queries. In the previous section we proposed a model to represent such a query database as RDF graphs. We now investigate the use of a native RDF query language, SPARQL 1.1, to access these graphs while keeping in mind the search and modification features presented in section 1. The following meta-querying scenarios apply:

1. finding a query knowing its (partial) structure;
2. finding queries targeting known base tables;
3. finding similar queries to a query being built;
4. completing a query structure given frequent structure patterns;
5. iteratively modifying queries e.g. to relax constraints.

Items 1, 2 and 3 can be categorized as read-only meta-querying, while items 4 and 5 involve writing query modifications. We look at these two approaches in the following subsections.

4.1 Query Search

Meta-querying scenarios that involve finding SQL queries similar to a given query, query template or query part can be reduced to authoring an appropriate SPARQL SELECT query. As our ASTs are modelled using RDF closed lists, retrieving one of its node by specifying some of its content is not sufficient: we have to specify the whole list by extension. For instance, finding SQL queries that perform a sum of an unknown column over Customers can be done with the SPARQL query S1.

```
1 SELECT ?q WHERE {
2  ?q a sql:Query ;
3   sql:argument (
4      [ sql:argument ([] [ a sql:Sum ] ) ]
5      [] []
6      [ a sql:GroupBy ;
7        sql:argument (
8          [ a sql:Column ; sql:identifier "Customer"] ) ] ) . }
```

Listing 3: SPARQL Query S1

However, this meta-query suffers silence, i.e. it will not return true positives that do not exactly have the three clauses (lines 4 and 5) preceding the `GROUP BY` one, neither those where the sum function is not located on the second position of

the select clause (line 3). We therefore use the SPARQL 1.1 property paths to recursively explore lists and look for specific SQL query parts, while releasing the constraints on irrelevant parts of the graph pattern. This leads to the query S2, which releases the writing constraint on terms position.

```
1 SELECT ?q WHERE {
2  ?q a sql:Query ;
3   (sql:argument/rdf:rest*/rdf:first)+
4      [ a sql:GroupBy ;
5        (sql:argument/rdf:rest*/rdf:first)+
6          [ a sql:Column ; sql:identifier "Customer"] ] ;
7   (sql:argument/rdf:rest*/rdf:first)+
8      [ a sql:Select ;
9        (sql:argument/rdf:rest*/rdf:first)+ [a sql:Sum] ] . }
```

Listing 4: SPARQL Query S2

Finally, we leverage RDFS entailment of the model described in section 3, in order to take properties and classes subsumption into account. The query S3 generalises S2 by looking for all aggregation functions instead of the specific sum, and reduces verbosity by replacing list explorations property paths with `sql:tree+`.

```
1 SELECT ?q WHERE {
2  ?q a sql:Query ;
3   sql:tree+ [ a sql:GroupBy ;
4        sql:tree+ [ a sql:Column ; sql:identifier "Customer"] ] ;
5   sql:tree+ [ a sql:Select ; sql:tree+ [a sql:AggregationFunction] ] . }
```

Listing 5: SPARQL Query S3

4.2 Query Modification

When a query yields insufficient or no results one might want to approximate it, i.e. to modify it in order to improve any combination of precision, recall or performance. For instance, relaxing the graph pattern constraints of a SPARQL query in order to make the homomorphism less restrictive [6]. Examples of query relaxation for SPARQL include replacing literal with variables, relaxing FILTER clauses with respect to values ranges, moving graph patterns to OPTIONAL clauses, or generalizing types with RDFS supertypes. Modification is also useful for comparison, when a syntactic similarity cannot be established between two queries, a search engine can apply rewriting rules in order to esablish semantic similarity. While most exploration or navigation languages cannot perform rewriting as they are by design limited to read-only operations, this can be done in SPARQL 1.1 with CONSTRUCT queries. We perform SQL query modification while only manipulating SPARQL queries, as described in the following approach. Given the target graph T of an SQL query Q modelled in RDF, we run a SPARQL CONSTRUCT query against T in order to produce the RDF graph M of the modified version Q' of Q. When serialized in Turtle [1], M can be directly reused as a SPARQL query graph pattern in a SELECT query to search for Q' into an arbitrary RDF store. This can be summarized in the scenario shown in listing 6.

```
1 let T // initial SQL query
2 let GP = eval( CONSTRUCT { M } WHERE { T } ) // modify T according to M
3 R = eval ( SELECT * WHERE { GP } ) // search for M query pattern
```

Listing 6: Query Modification in SPARQL

It is possible to apply iteratively as many CONSTRUCT queries as required, for instance until the result set R provides a required amount of answers. This approach is natively implemented in the Corese/KGRAM 3.11 RDF toolkit [3], as its engine uses the same abstraction for both RDF and query graphs: one can cast the result set object of a CONSTRUCT query and use it as a graph pattern for a programmatically generated SELECT query.

5 Conclusion & Perspectives

Dynamic documents populated by SQL queries are predominant in enterprise environments. Understanding these queries can improve IR systems that deal with these documents, and create a common metadata layer among heterogeneous documents. We introduced an approach to represent SQL queries with RDF graphs, providing an RDFS vocabulary that can capture SQL query statements. We presented initial SPARQL 1.1 queries that allow to search the queries by looking at their content and structure, and a generic query modification technique that can serve a query search engine. Finally, our approach can be considered as a way to produce linked enterprise data over repositories of disparate dynamic documents that only share SQL queries, thus providing linked query data to supplement proprietary metadata.

Future work will involve practical problems investigation. In particular, URI minting for relational databases objects should provide a basis to connect ASTs of queries that share references to identical tables and columns. Second, the query modification problem brings many challenges that depend on the list-based tree modelling. We will further evaluate whether SPARQL 1.1 provides the necessary constructs to perform ad-hoc tree manipulation while keeping queries concise and intuitive. In particular, we plan to compare the use of CONSTRUCT with the SPARQL Update clauses INSERT and DELETE. Tree manipulation operations of interest include tree pruning, subtree addition and moving, connection. On the use case perspective, we are implementing a proof of concept that analyzes documents repositories and builds the RDF graphs representing their SQL queries with intermediary ASTs. We will use these as target graphs to evaluate the SPARQL meta-queries that compute similarity measures, and to formally define equivalence relations that define the notion of query similarity in the context of BI documents. Finally, we plan to extend or refactor the query model in order to evaluate the approach with other languages that produce ASTs, such as multidimensional expressions (MDX).

References

1. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language. Team Submission, W3C (March 2011), http://www.w3.org/TeamSubmission/turtle/
2. Van den Bussche, J., Vansummeren, S., Vossen, G.: Towards practical meta-querying. Information Systems 30(4), 317–332 (June 2005)
3. Corby, O., Faron-Zucker, C.: The KGRAM Abstract Machine for Knowledge Graph Querying. In: Web Intelligence. pp. 338–341 (2010)
4. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Working Draft, W3C (May 2012), http://www.w3.org/TR/r2rml/
5. Ferré, S.: Extension du langage de requêtes LISQL pour la représentation et l'exploration d'expressions mathématiques en RDF. In: IC 2012, 23èmes Journées Francophones d'Ingénierie des Connaissances. Paris, France (June 2012)
6. Hurtado, C., Poulovassilis, A., Wood, P.: Query Relaxation in RDF. In: Spaccapietra, S. (ed.) Journal on Data Semantics X, Lecture Notes in Computer Science, vol. 4900, pp. 31–61. Springer Berlin / Heidelberg (2008)
7. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: SnipSuggest: Context-Aware Autocompletion for SQL. Proc. VLDB Endow. 4, 22–33 (October 2010)
8. Knublauch, H.: SPIN - SPARQL Syntax. Member Submission, W3C (February 2011), http://www.w3.org/Submission/spin-sparql/
9. Konstantinidis, G., Ambite, J.L.: Scalable Query Rewriting: A Graph-Based Approach. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 97–108. SIGMOD '11, ACM, New York, NY, USA (2011)
10. Priebe, T., Pernul, G.: Towards integrative enterprise knowledge portals. In: Proceedings of the twelfth international conference on Information and knowledge management. pp. 216–223. CIKM '03, ACM, New York, NY, USA (November 2003)